# A Web Scraping Tutorial using R.

## Author Note

Alex Bradley, and Richard J. E. James, School of Psychology, University of Nottingham

AB is the guarantor. AB created the website and videos. Both authors drafted the manuscript. All authors read, provided feedback and approved the final version of the manuscript.

The authors declare that they have no conflicts of interest pertaining to this manuscript.

Correspondence concerning this article should be addressed to Alex Bradley, School of Psychology, University of Nottingham, Nottingham, NG7 2RD. Email alexander.bradley@nottingham.ac.uk. Tel: 0115 84 68188.

**Abstract**

The ubiquitous use of the internet for all manner of daily tasks means that there are now large reservoirs of data that can provide fresh insights into human behaviour. One of the key barriers preventing more researchers from utilising online data is that researchers do not have the skills to access the data. This tutorial aims to address this gap by providing a practical guide to web scraping online data using the popular statistical language *R*. Web scraping is the process of automatically collecting information from websites, which can take the form of numbers, text, images or videos. In this tutorial, readers will learn how to download web pages, extract information from those pages, be able to store the extracted information and learn how to navigate across multiple pages of a website. A website has been created to assist readers in learning how to web scrape. The website contains a series of examples that look at how to scrape a single web page, and how to scrape multiple webpages. Each example is accompanied by a video describing the process of web scraping and an example *R* script. Accompanying exercises are also provided to help readers practice their knowledge and skills.

**Keywords**: Web scraping, web crawling, reverse engineering, big data, open science.

Abstract Word Count: 200

Total Word Count: 2,987

So many behaviours and interactions now happen and are stored online presenting researchers with a wealth of data to help improve our understanding of human behaviour. For example, helping us predict people political preferences (Ceron, Curini, Iacus, & Porro, 2014; Malouf & Mullen, 2015), understand motivation behind charitable donations to crowd fundraising campaigns(Agrawal, Catalini, & Goldfarb, 2015; Kuppuswamy & Bayus, 2018), or even which products people tend to compare before buying (Feldman, Fresko, Goldenberg, Netzer, & Ungar, 2007). Yet psychology has been slow to utilise online data. One of the main barriers preventing psychologists from using online data is a skills gap (Adjerid & Kelley, 2018; Paxton & Griffiths, 2017). The underlying issue within this skills gap is that new data methods like web scraping require a knowledge of programming that most psychologists have not been trained to perform (Adjerid & Kelley, 2018). The aim of this tutorial is to address this skills gap by giving a practical hands-on guide to web scraping using *R*.

Web scraping allows the rapid collection and processing of a large amount of data from online sources which can be numerical, textual or a collection of images/videos (Marres & Weltevrede, 2013). Web scraping is time efficient allowing thousands of data points to be automatically collected whereas before this would have involved painstaking manual effort. Web scraping is, therefore, less labour intensive, faster and less open to human error than the traditional copy and pasting method (Nylen & Wallisch, 2017). Web scraping also has the advantage that researchers are able to acquire novel, untouched datasets without the need for research grants to fund the purchasing of expensive equipment or incurring the costs of participant payments. In this tutorial, we cover how to download a web page, how to extract information from the downloaded page, how to store extracted information and finally, how to move across webpages on a website.

**Disclosures**

A website containing the examples, exercises, and videos for this tutorial can be found at https://practicewebscrapingsite.wordpress.com/. All the example *R* scripts and PowerPoints used in the videos can be downloaded from the Open Science Framework at https://osf.io/6hmqg/.

**Practice Web Scraping Site**

The website was specifically designed to help you learn about the process of web scraping and to provide a safe environment for you to practice web scraping. There is an introductory video that provides an overview of web scraping, the tools that we will be using to webscrape and good practices whilst web scraping. The website contains four main examples each accompanied by practice exercises. Each example is accompanied by an *R* script and video explaining each of the steps involved in the web scraping process. In example 1, readers will learn how to download, extract and store information from a single webpage. Examples 2 and 3, explain how to download, extract and store information whilst using links built into the website to move across multiple webpages. Example 4 shows readers how to download, extract and store information whilst moving across webpages by manipulating web addresses (URL's). We would encourage users after watching the example video, whilst following along with the example *R* script, to then take the time to complete the accompanying exercises before moving on to the next example.

**Learning Objective and Assumed Knowledge**

The learning objectives of this tutorial are to teach readers how to automatically collect information from a website. In particular, readers should after this tutorial able to download a webpage, know how to extract information from that web page, be able to store extracted information and understand different methods of moving from page to page whilst web scraping. An understanding of *R* and RStudio would be helpful but is not required. The tutorial has been designed so that novices to web scraping and those with little to no programming experience find the material accessible and can begin to develop their own scraping skills. Those who already have *R* and RStudio installed and have a basic understanding of the *R* language may wish to skip to the four steps involved in web scraping.

**Installation of *R*, RStudio and SelectorGadget.**

Before we can learn how to web scrape we first need to download and install the programs we are going to use. All of these tools are free to download and use. First, you will need to download and install *R* from https://cran.rstudio.com/. Second, we would recommend download and installing RStudio from https://www.rstudio.com/. All the code for the rest of this tutorial will be run in the

script window of RStudio. You can create new scripts in RStudio by clicking on File > New File > R Script. Finally, we will need SelectorGadget which can be downloaded at https://selectorgadget.com/. If you do not use Chrome as your web browser you will also need to download this first (https://www.google.com/chrome/) and then download Selectorgadget. For more information about how to download these programs, a brief introduction to web scraping and an overview of the website please watch our introduction to web scraping video (https://practicewebscrapingsite.wordpress.com/).

**Packages and Functions in R**

*R* is an incredibly versatile programming language capable of performing many different tasks like web scraping, statistical analyses, data visualisations etc. The reason for its versatility is it has a large community of users that create software, in the form of packages, that other users can use. A package is a collection of functions designed to perform a task. For example, in this tutorial, we use the "rvest" package which contains a variety of functions that will help us to web scrape. A function is a technical term to denote code that modifies or manipulates some input to produce the desired output. For example, to calculate the mean we could use the mean function by providing it with a vector (column) of numbers (i.e. `mean(numbers)`). The function often takes additional instructions, known as arguments, that adjust how it modifies or manipulates the input. For example, we could modify the mean function by specifying whether we want missing values to be included (True or False) by using the na.rm argument (i.e. `mean(numbers, na.rm = TRUE)`). In order to use functions contained in a package you first need to install and load that package.

**Installing and Loading *R* packages**

Downloading and installing packages for use in *R* is a two-step process. First one needs to download the package by using the function `install.packages("package name")` and second to load it into *R* by using the function `library(package name)`.

For the rest of this tutorial, you will need to download and install the rvest package by typing and running the following code (see Figure *1*)(Wickham, 2016).

```
install.packages("rvest")# This only needs to be called once after
#downloading R and Rstudio's. The package will be stored for future
#sessions.

library(rvest) #This command needs to be run at the beginning of
#each new session in RStudio's.
```

*Figure 1.R code showing how to download and install the rvest package.*

Note that once you have installed a package you will never need to download it again, however, every time you start a new session of RStudio you will need to run the library function.

**Four key steps involved whilst web scraping**

**How to download a web page.**

To download a web page we use the `read_html` function and supply it with the URL of the webpage we wish to scrape (i.e. `read_html("address of website")`). In example 1 on the website, we wish to collect the title, main text and picture links for three articles stored on a single webpage. In order, to download this page we type in and run the code in Figure *2*.

```
Example1 <-
read_html("https://practicewebscrapingsite.wordpress.com/example-1/")
```

*Figure 2. Code illustrating how to download a webpage.*

The `read_html` function downloads the webpage at the address given and then assigns that information using the less than and hype notation (<-) to an object called Example1. In technical terms, Example1 is what we call a Document Object Model because it holds all the data of the webpage and preserves the structure of the information held on that webpage. A good analogy to this is to imagine taking a physical book and converting the information into a digital book available on your e-reader. The book on the ereader will still contain the same number of chapters, pages and paragraphs with the same text in each of those sections. The `read_html` function mirrors the digital book on the ereader by collecting and preserving the structure of the information held on the webpage. This is important as allows us in the next step to extract just the information we are interested in from

the sections of the webpage that contain that information.

**How to extract information from a webpage.**

Extracting information from a webpage involves two steps. In the first step, we locate the information that we wish to collect on the webpage and in the second step we specify what information at that location we wish to extract. A good analogy for this is using a textbook to find a famous quote by an author. In step one, you turn to the chapter and page number where that author is mentioned and in step two you copy the famous words by the author.

To do step one we use the `html_nodes` function and provide two additional pieces of information: the object holding the downloaded webpage and the address to the information we wish to extract (i.e. `html_nodes(webpage, "address to information")`). In order, to generate the address to the information we want, we use Selectorgadget (see Figure 3 for an explanation of how this is achieved). By the end of step one, we have used the `html_nodes` function to locate the relevant part of the website where the information we wish to extract is stored. This information is then passed on to step two using the pipe operator (`%>%`).

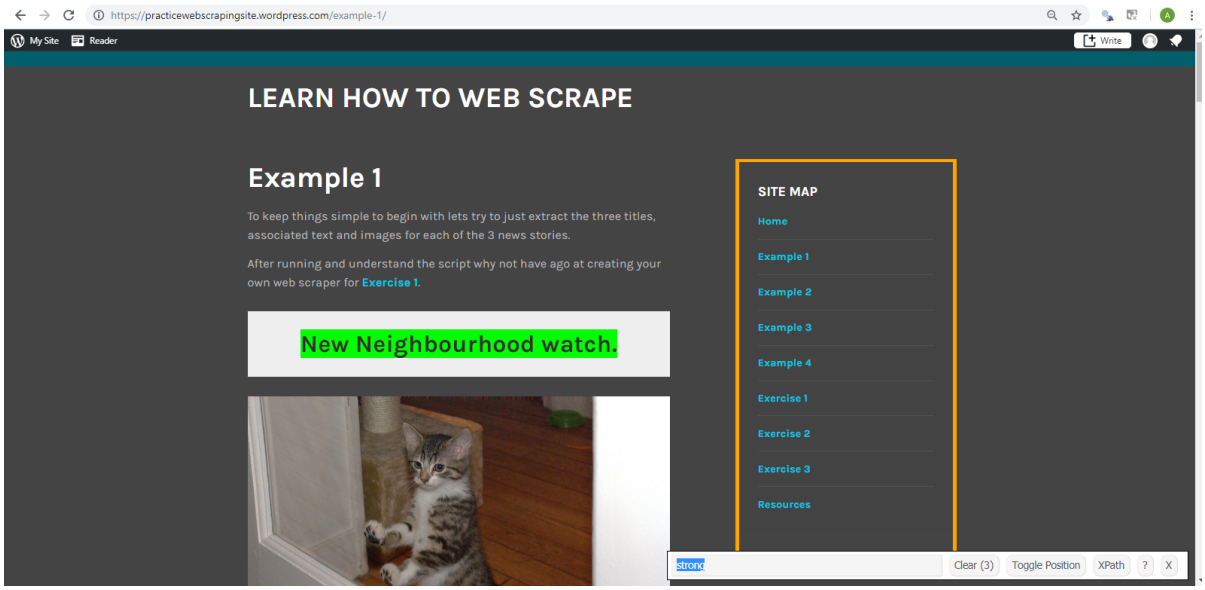


*Figure 3. Once you have opened Chrome and installed SelectorGadget there will be an icon in the top right that looks like this*  *. Click on this to open SelectorGadget. Then select the information that you wish to extract from the webpage for example the title of the article. Look down the page and make sure that only the information you wish to extract is highlighted green or yellow. If additional information that is not required is highlighted yellow or green click on that to unselect it. When only the right information is highlighted green or yellow then copy and paste the address Selectorgadget*

*generates into the html_nodes command. In this example, the titles on the page can be collected using the address "strong".*

In step two we use one of three commands depending upon the type of information we wish to extract. If we wish to extract text we use the `html_text` function (i.e.`html_text()`). If we want to extract links from the webpage we use the `html_attr` function with the additional href argument (i.e. `html_attr("href")`). Or we can collect the address of images to download later by using the `html_attr` function with additional argument src (i.e. `html_attr("src")`).

Figure *4* shows how in Example 1 we extract the titles, text and address to the pictures of three articles stored on the webpage. For a live demonstration of how to extract information please see the video on the website entitled "Example 1: Scraping a single webpage".

```
#The pipe operator (%>%) takes output from one function and passes
#it to another without the need to store the output in between the
#functions. For example, below the output from html_nodes is passed
#on to the function html_text().
Title <- html_nodes(Example1, "strong") %>% html_text()
Text <- html_nodes(Example1, ".Content") %>% html_text()
Images <- html_nodes(Example1, "#post-25 img") %>% html_attr("src")
```

*Figure 4. Code showing how to extract the title, main text and image from Example 1 using* the *`html_nodes`, `html_text` or `html_attr` commands.*

**How to store information collected whilst web scraping.**

There are several ways that we could choose to store information like saving it in a database or storing it in vectors (like a column of data). The best approach to storing information will depend upon the type of data you are extracting and the amount of data you are collecting. For simplicity, in this tutorial, we store information in vectors. This process changes depending upon whether you are scraping a single page or multiple pages.

We shall begin by explaining a single page using the Example 1 code presented in Figure *4*. The three titles on the page are extracted by the `html_nodes` and `html_text` command. This information is then assigned to the vector called "Title".

Storing information when web scraping over multiple pages is a little more complicated

because as we move over each webpage extracting information and storing it to a vector when we do the same for the next page the information captured from the first page will be overwritten. To avoid this problem, we use the following three-step process. First, we initialise an empty vector called Title (see Figure *5*). Second, we extract the information from a webpage using the `html_nodes` and `html_text` function which is stored in a vector called "Heading". Third, we add the information captured in "Heading" to the initially empty vector called "Title". As the web scraper goes over different pages the information in the Heading vector will be overwritten with new information that is then added to all the previously extracted headings stored in the Title vector. For a demonstration of this technique please watch the Example 2 video entitled "Example 2 scraping multiple web pages."

```
#First initialising an empty vectors to store information in:
Title <- c()
#Second extracting information:
Heading <- html_nodes(Example2, ".entry-title") %>% html_text()
#Third storing information:
Title <- c(Title,Heading)
```

*Figure 5. Extract of code used in Example 2 to store information.*

**How to move across multiple pages.**

There are a variety of methods to move across a website and often the way the website is designed will determine the approach to use. To keep things simple, this tutorial will outline two common approaches used by web scrapers to move across webpages: following links in a webpage to other pages and manipulating the web address.

To follow links on a webpage you first need to download a webpage containing links to all the other pages to be visited and then extract and store those links. For example, in Figure *6* the webpage is downloaded using the `read_html` function and stored in "Example2". The `html_nodes` and `html_attr` functions are used to extract the links which are then stored in "BlogPages". The next step is to use a "`for`" loop to iterate through the extracted links. A "`for`" loop is a way of repeating a block of code. In Figure *6*, we use a "`for`" loop to iterate over the

links stored in "BlogPages". The "i" becomes each one of the links stored in BlogPages and runs through the indented code held between the two curly brackets {}. Notice, "i" is passed to the read_html function to download this new webpage which we can then extract and store information from. It is the act of passing "i" to the read_html function that allows the scraper to navigate over multiple pages.

```
Title <- c() #Initialise the empty vector
#Download the webpage
Example2 <-
read_html("https://practicewebscrapingsite.wordpress.com/example-2/")
#Extract the links
BlogPages <- html_nodes(Example2, ".Links a") %>% html_attr("href")
#Using each of the links, stored in i,  to visit a webpage using a
#'for' loop.
for (i in BlogPages){
  #Code in here is repeated for every link extracted and stored in
  #BlogPages
  Example2 <- read_html(i) #Downloading the webpage
  Heading <- html_nodes(Example2, ".entry-title") %>% html_text()
#Extracting information
  Title <- c(Title,Heading) #Storing information
}
```

*Figure 6. An extract from Example 2 showing how to move across multiple pages of a website using the links stored on a webpage.*

To move over webpages by manipulating URLs we need to identify a part of the URL that systematically changes over the webpages we wish to scrape. We then need to artificially manipulate that URL to move over the different pages. To illustrate this Example 4 contains webpages where the URL changes by the page number specified i.e. https://practicewebscrapingsite.wordpress.com/example-4-page-0/ , https://practicewebscrapingsite.wordpress.com/example-4-page-1/. For this example, we need to generate a sequence of numbers to represent the different page numbers. In Figure 7, we use the seq function to generate a sequence of numbers starting from 0 and going up to 1 in increments of 1.

This information (`0,1`) is saved to "`Pages`". A `for` loop is then used to iterate over "Pages" with "i" becoming the page numbers `0` and `1`. The `paste` function is used to generate the URL by taking the part of the URL that does not change and adding that to the number stored in "`i`". This URL is then stored in `WebPageURL` and then passed to `read_html` function to download the new webpage. Information from this new webpage can then be extracted and stored.

```
#We generate a sequence of numbers by using the sequence function
#which takes the arguments seq(first number, last number, increment
#change)

Pages <- seq(0,1,1)

#Use a for loop to iterate over the first two pages. The i in the
#for loop will become 0 and 1 so the code within the for loop will
#run twice.

for (i in Pages){

Sys.sleep(2) #This function inserts a 2 second pause before carrying
#on with the rest of the code. This is really important to avoid
#putting undue stress on the website server which can lead to a
#web scraper getting banned from a website.

#Use the paste function to generate a unique URL by adding the main
#web address to the new page number held in i. The sep argument is
#left blank.

 WebpageURL <-
paste("https://practicewebscrapingsite.wordpress.com/example-4-page-
",i,"", sep="")

#Download the webpage using the new URL generated by the paste
#function.

  Example4 <- read_html(WebpageURL)

  #Code to extract the information from each page inserted here

  #Code to Store information inserted here.

}
```

*Figure 7. Extract of code from Example 4 where we manipulate URLs to navigate over multiple webpages.*

## Good practices and the ethics of web scraping

Before scraping a website it is a good idea to check if they offer an Application Program Interface (API) which allow users to quickly collect data directly from the database behind the website. If they do offer an API that contains the information you need it would be easier to use the API. When web

scraping it is a good idea to insert pauses between downloading webpages as this helps spread out the traffic to the website. In Figure 7, we use the `Sys.sleep` function to insert two-second delays before downloading the next webpage. Before starting your own web scraping project it would be a good idea to check what your institutional review board policy is on web scraping it might be the case that you need to make an ethics application or it might be classified as archival data without the need for an ethics application. As a general rule of thumb, any information stored behind a username and password is considered private and ought not to be web scraped. Finally, the methods presented here should help you to scrape many websites on the internet however, there may be some sites that display information in unusual formats which may make them more difficult to scrape. It is worth checking that you can download and extract information from a single page before building a complete web scraper.

## Summary of Content

In this tutorial, we have introduced readers to what web scraping is and why it is a useful data collection tool for psychologists to learn. The concept of packages and functions in *R* was introduced, as well as, an explanation of how to download and install *R* packages ready for use. Readers should feel confident in their ability to conduct the four key steps of web scraping: downloading webpages, extracting information from downloaded pages, storing that extracted information and then using either web links or manipulating URLs to navigate across multiple web pages. We strongly recommend that readers work through the examples provided on the accompanying website to further build their knowledge of web scraping and gain more experience of web scraping by completing the exercises on the website.

## Declaration of Conflicting Interests.

The authors declare that there were no conflicts of interest with respect to the authorship or the publication of this article.

## References

Adjerid, I., & Kelley, K. (2018). Big Data in Psychology: A Framework for Research Advancement. *American Psychologist*. https://doi.org/10.1037/amp0000190

Agrawal, A., Catalini, C., & Goldfarb, A. (2015). Crowdfunding: Geography, Social Networks, and the Timing of Investment Decisions. *Journal of Economics and Management Strategy*, *24*(2), 253–274. https://doi.org/10.1111/jems.12093

Ceron, A., Curini, L., Iacus, S. M., & Porro, G. (2014). Every tweet counts? How sentiment analysis of social media can improve our knowledge of citizens' political preferences with an application to Italy and France. *New Media and Society*, *16*(2), 340–358. https://doi.org/10.1177/1461444813480466

Feldman, R., Fresko, M., Goldenberg, J., Netzer, O., & Ungar, L. (2007). Extracting product comparisons from discussion boards. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 469–474. https://doi.org/10.1109/ICDM.2007.27

Kuppuswamy, V., & Bayus, B. (2018). Crowdfunding Creative Ideas: The Dynamics of Project Backers. In D. Cumming & L. Hornuf (Eds.), *The Economics of Crowdfunding* (1st ed., Vol. 1, pp. 151–182). Palgrave Macmillan. https://doi.org/10.1017/CBO9781107415324.004

Malouf, R., & Mullen, T. (2015). Taking sides: user classification for informal online political discourse. *Internet Research*, *18*(2), 177–190.

Marres, N., & Weltevrede, E. (2013). Scraping the Social? *Journal of Cultural Economy*, *6*(3), 313–335. https://doi.org/10.1080/17530350.2013.772070

Nylen, E. L., & Wallisch, P. (2017). Web Scraping. *Neural Data Science*, 277–288. https://doi.org/10.1016/B978-0-12-804043-0.00010-6

Paxton, A., & Griffiths, T. L. (2017). Finding the traces of behavioral and cognitive processes in big data and naturally occurring datasets. *Behavior Research Methods*, *49*(5), 1630–1638. https://doi.org/10.3758/s13428-017-0874-x

Wickham, H. (2016). rvest manual. Retrieved from https://cran.r-project.org/web/packages/rvest/rvest.pdf